# Web技术

## OSI7层模型与TCP/IP协议栈4层

### OSI

应用层   
1.主要功能 ：用户接口、应用程序   
application 2.典型设备：网关   
3.典型协议、标准和应用：TELNET, FTP, HTTP   
  
表示层   
1.主要功能 ：数据的表示、压缩和加密   
presentation2.典型设备：网关   
3.典型协议、标准和应用：ASCLL、PICT、TIFF、JPEG、 MIDI、MPEG   
  
会话层   
1.主要功能 ：会话的建立和结束   
session2.典型设备：网关   
3.典型协议、标准和应用：RPC、SQL、NFS 、X WINDOWS、ASP   
  
传输层   
1.主要功能 ：端到端控制   
transport 2.典型设备：网关   
3.典型协议、标准和应用：TCP、UDP、SPX   
  
网络层   
1.主要功能 ：路由，寻址   
network2.典型设备：路由器   
3.典型协议、标准和应用：IP、IPX、APPLETALK、ICMP   
  
数据链路层   
1.主要功能 ：保证误差错的数据链路   
data link 2.典型设备：交换机、网桥、网卡   
3.典型协议、标准和应用：802.2、802.3ATM、HDLC、FRAME RELAY   
  
物理层   
1.主要功能 ：传输比特流   
physical2.典型设备：集线器、中继器   
3.典型协议、标准和应用：V.35、EIA/TIA-232

**TCP/IP:**  
数据链路层：ARP(ip->mac),RARP(mac->ip)  
网络层： IP,ICMP(用于在[主机](http://baike.baidu.com/view/23880.htm" \t "_blank)、路由器之间传递控制消息。控制消息是指[网络通](http://baike.baidu.com/view/8079702.htm)不通、主机是否可达、[路由](http://baike.baidu.com/view/18655.htm)是否可用等网络本身的消息，典型应用就是**ping**),IGMP(用于管理因特网协议[*多播*](https://zh.wikipedia.org/wiki/%E5%A4%9A%E6%92%AD)组成员的一种通信协议)  
传输层：TCP ,UDP  
应用层：Telnet(远程登录),FTP(文件传输协议),SMTP([简单邮件传输协议](http://baike.baidu.com/view/576460.htm)),SNMP(简单网络管理协议). POP3(即“[邮局协议](http://baike.baidu.com/view/544380.htm)版本3”)

### url到页面的过程

**1. DNS域名解析**

在浏览器DNS缓存中搜索🡪在操作系统DNS缓存中搜索🡪读取系统hosts文件，查找其中是否有对应的ip🡪向本地配置的首选DNS服务器发起域名解析请求

**2. 建立TCP连接**

为了准确地传输数据，TCP协议采用了三次握手策略。

**3. 发起HTTP请求**

**请求方法：**

**Get：获取资源；post：传输实体主题**

**4. 接受响应结果**

状态码：



**5. 浏览器解析html**

浏览器按顺序解析html文件，构建DOM树，在解析到外部的css和js文件时，向服务器发起请求下载资源，若是下载css文件，则解析器会在下载的同时继续解析后面的html来构建DOM树，在下载js文件和执行它时，解析器会停止对html的解析。

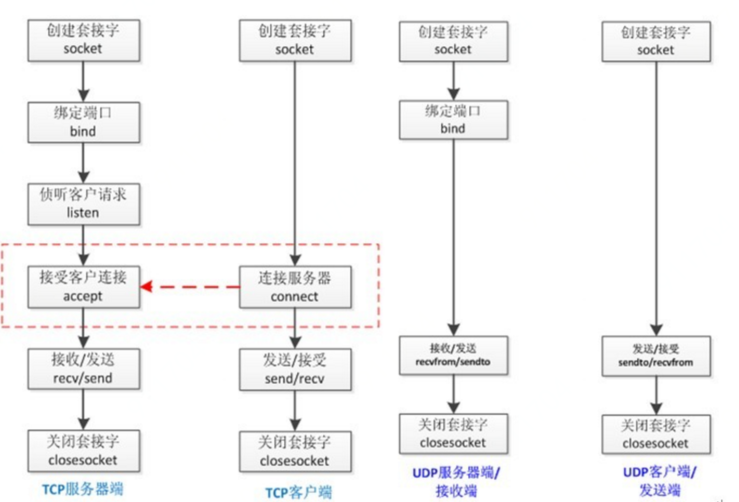
**6. 浏览器布局渲染**

## TCP/IP

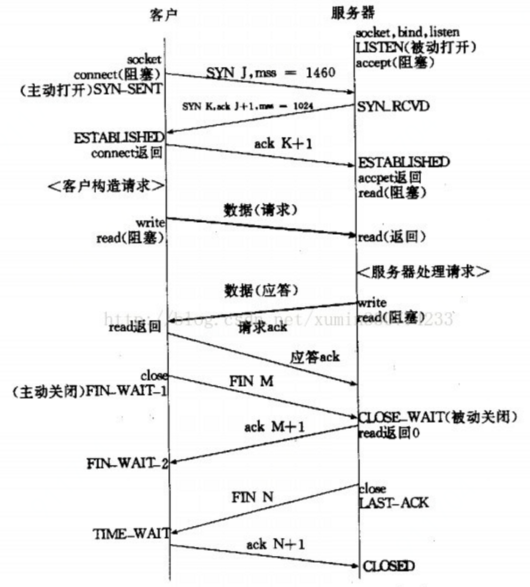
### TCP与UDP比较



**TCP/UDP编程模型：**



### 三次握手、四次挥手



**为什么连接的时候是三次握手，关闭的时候却是四次握手？**

因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

**为什么不能用两次握手进行连接？**

3次握手完成两个重要的功能，既要双方做好发送数据的准备工作(双方都知道彼此已准备好)，也要允许双方就初始序列号进行协商，这个序列号在握手过程中被发送和确认。

现在把三次握手改成仅需要两次握手，死锁是可能发生的。作为例子，考虑计算机S和C之间的通信，假定C给S发送一个连接请求分组，S收到了这个分组，并发送了确认应答分组。按照两次握手的协定，S认为连接已经成功地建立了，可以开始发送数据分组。可是，C在S的应答分组在传输中被丢失的情况下，将不知道S 是否已准备好，不知道S建立什么样的序列号，C甚至怀疑S是否收到自己的连接请求分组。在这种情况下，C认为连接还未建立成功，将忽略S发来的任何数据分 组，只等待连接确认应答分组。而S在发出的分组超时后，重复发送同样的分组。这样就形成了死锁。

**TIME\_WAIT的意义（为什么要等于2MSL）**

MSL是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃。为了确认最后一个ack包对方可以收到，必须要等待2msl（包往返时间），因为对方在超时后将重发fin包，这时等待端可以重新发送ack包使对方正确关闭。

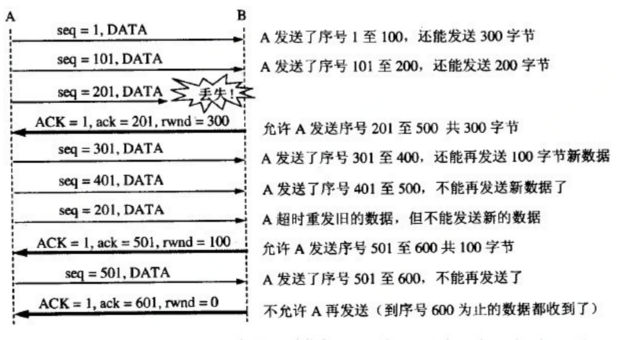
### 流量控制与拥塞控制

**TCP的流量控制**

1. 利用滑动窗口实现流量控制

如果发送方把数据发送得过快，接收方可能会来不及接收，这就会造成数据的丢失。所谓**流量控制**就是让发送方的发送速率不要太快，要让接收方来得及接收。

设A向B发送数据。在连接建立时，B告诉了A：“我的接收窗口是 rwnd = 400 ”(这里的 rwnd 表示 receiver window) 。因此，发送方的发送窗口不能超过接收方给出的接收窗口的数值。请注意，TCP的窗口单位是字节，不是报文段。TCP连接建立时的窗口协商过程在图中没有显示出来。再设每一个报文段为100字节长，而数据报文段序号的初始值设为1。大写ACK表示首部中的确认位ACK，小写ack表示确认字段的值ack。



 Nagle算法：若发送应用进程把要发送的数据逐个字节地送到TCP的发送缓存，则发送方就把第一个数据字节先发送出去，把后面到达的数据字节都缓存起来。当发送方接收对第一个数据字符的确认后，再把发送缓存中的所有数据组装成一个报文段再发送出去，同时继续对随后到达的数据进行缓存。只有在收到对前一个报文段的确认后才继续发送下一个报文段。当数据到达较快而网络速率较慢时，用这样的方法可明显地减少所用的网络带宽。Nagle算法还规定：当到达的数据已达到发送窗口大小的一半或已达到报文段的最大长度时，就立即发送一个报文段。

**糊涂窗口综合证：**TCP接收方的缓存已满，而交互式的应用进程一次只从接收缓存中读取1字节（这样就使接收缓存空间仅腾出1字节），然后向发送方发送确认，并把窗口设置为1个字节（但发送的数据报为40字节的的话）。接收，发送方又发来1个字节的数据（发送方的IP数据报是41字节）。接收方发回确认，仍然将窗口设置为1个字节。这样，网络的效率很低。要解决这个问题，可让接收方等待一段时间，使得或者接收缓存已有足够空间容纳一个最长的报文段，或者等到接收方缓存已有一半空闲的空间。只要出现这两种情况，接收方就发回确认报文，并向发送方通知当前的窗口大小。此外，发送方也不要发送太小的报文段，而是把数据报积累成足够大的报文段，或达到接收方缓存的空间的一半大小。

**TCP的拥塞控制**

拥塞：即对资源的需求超过了可用的资源。若网络中许多资源同时供应不足，网络的性能就要明显变坏，整个网络的吞吐量随之负荷的增大而下降。

拥塞控制：**防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。**拥塞控制所要做的都有一个**前提：网络能够承受现有的网络负荷。**拥塞控制是一个**全局性的过程**，涉及到所有的主机、路由器，以及与降低网络传输性能有关的所有因素。

流量控制：**指点对点通信量的控制**，是端到端的问题。流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

**慢开始和拥塞避免：**

慢开始算法：在刚刚开始发送报文段时，先把拥塞窗口 cwnd 设置为一个最大报文段MSS的数值。而在每收到一个对新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。用这样的方法逐步增大发送方的拥塞窗口 cwnd ，可以使分组注入到网络的速率更加合理。每经过一个传输轮次，拥塞窗口 cwnd 就加倍。一个传输轮次所经历的时间其实就是往返时间RTT。不过“传输轮次”更加强调：把拥塞窗口cwnd所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。

为了防止拥塞窗口cwnd增长过大引起网络拥塞，还需要设置一个慢开始门限ssthresh状态变量（如何设置ssthresh）。慢开始门限ssthresh的用法如下：

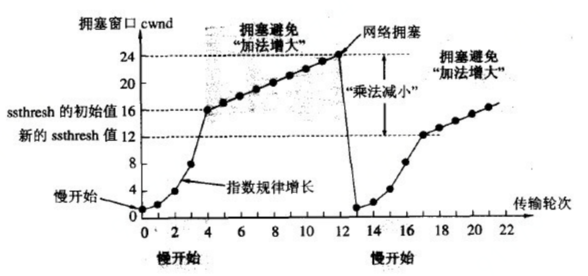
    当 cwnd < ssthresh 时，使用上述的慢开始算法。

    当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。

当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞控制避免算法。

拥塞避免算法：让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。这样拥塞窗口cwnd按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。

无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认），就要把慢开始门限ssthresh设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。



**快重传和快恢复**

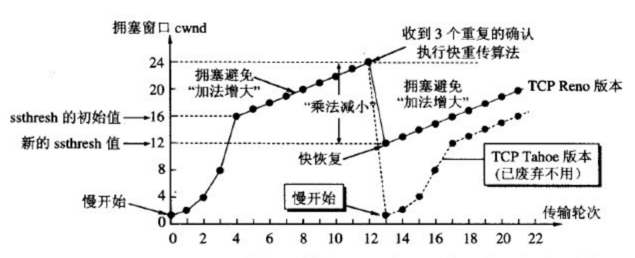
如果发送方设置的超时计时器时限已到但还没有收到确认，那么很可能是网络出现了拥塞，致使报文段在网络中的某处被丢弃。这时，TCP马上把拥塞窗口 cwnd 减小到1，并执行慢开始算法，同时把慢开始门限值ssthresh减半。这是不使用快重传的情况。

快重传算法首先要求接收方每收到一个的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时才进行捎带确认。发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段。

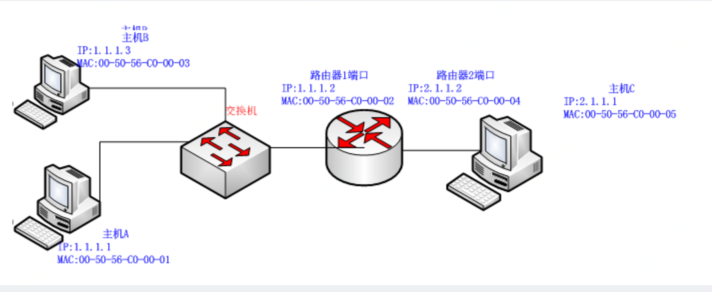
与快重传配合使用的还有快恢复算法，其过程有以下两个要点：

 <1>. 当发送方连续收到三个重复确认，就执行“乘法减小”算法，把慢开始门限ssthresh减半。这是为了预防网络发生拥塞。请注意：接下去不执行慢开始算法。

<2>. 由于发送方现在认为网络很可能没有发生拥塞，因此与慢开始不同之处是现在不执行慢开始算法（即拥塞窗口cwnd现在不设置为1），而是把cwnd值设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。



### Ping命令过程详解



**同一网段内：**

首先，如果主机A，要去ping主机B，那么主机A，就要封装二层报文，他会先查自己的MAC地址表，如果没有B的MAC地址，就会向外发送一个ARP广播包。

交换机会收到这个报文后，交换机有学习MAC地址的功能，所以他会检索自己有没有保存主机B的MAC地址，如果有，就返回给主机A，如果没有，就会向所有端口发送ARP广播，其它主机收到后，发现不是在找自己，就纷纷丢弃了该报文，不去理会。直到主机B收到了报文后，就立即响应，我的MAC地址是多少，同时学到主机A的MAC地址，并按同样的ARP报文格式返回给主机A。

这时候主机A学到了主机B的MAC地址，就把这个MAC地址封装到ICMP协议的二层报文中向主机B发送。当主机B收到了这个报文后，发现是主机A 的ICPM回显请求，就按同样的格式，返回一个值给主机A，这样就完成了同一网段内的ping过程。

**不同网段内：**

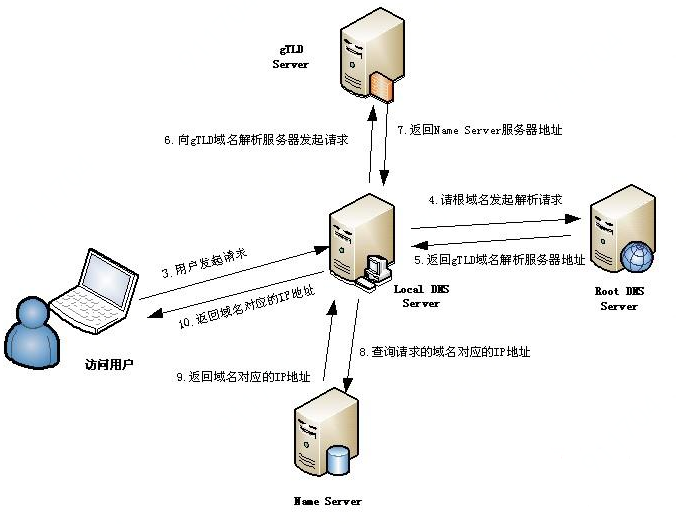
如果主机A要ping主机C，那么主机A发现主机C的IP和自己不是同一网段，他就去找网关转发，但是他也不知道网关的MAC地址情况下呢？他就会向之前那个步骤一样先发送一个ARP广播，学到网关的MAC地址，再发封装ICMP报文给网关路由器.。

当路由器收到主机A发过来的ICMP报文，如果网关也没有主机C的MAC地址，还是要向前面一个步骤一样，ARP广播一下即可相互学到。

在主机C已学到路由器2端口MAC地址，路由器2端口转发给路由器1端口，路由1端口学到主机A的MAC地址的情况下，他们就不需要再做ARP解析，就将ICMP的回显请求回复过来。

如果是ping域名，需要先进行域名解析，即dns。

### Dns域名解析过程



1. 浏览器会检查缓存中有没有这个域名对应的解析过的IP地址，如果缓存中有，这个解析过程就结束。
2. 如果用户浏览器缓存中没有数据，浏览器会查找操作系统缓存中是否有这个域名对应的DNS解析结果。（hosts文件）
3. 前两个过程无法解析时，就要用到我们网络配置中的"DNS服务器地址"了。操作系统会把这个域名发送给这个LDNS，也就是本地区的域名服务器。
4. 如果LDNS仍然没有命中，就直接到Root Server域名服务器请求解析。
5. 根域名服务器返回给本地域名服务器一个所查询的主域名服务器（gTLD Server）地址。gTLD是国际顶级域名服务器，如.com、.cn、.org等，全球只有13台左右。
6. 本地域名服务器LDNS再向上一步返回的gTLD服务器发送请求。
7. 接受请求的gTLD服务器查找并返回此域名对应的Name Server域名服务器的地址，这个Name Server通常就是用户注册的域名服务器。
8. Name Server域名服务器会查询存储的域名和IP的映射关系表，在正常情况下都根据域名得到目标IP地址。
9. 把解析的结果返回给用户，用户根据TTL值缓存在本地系统缓存中，域名解析过程结束。

## HTTP

设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。是用于从WWW服务器传输超文本到本地浏览器的传输协议。默认使用80端口，

### http1.0/http1.1/http2.0

#### HTTP1.0

HTTP 1.0规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个TCP连接，服务器完成请求处理后立即断开TCP连接，服务器不跟踪每个客户也不记录过去的请求。http1.0被抱怨最多的就是**连接无法复用**，和**head of line blocking**这两个问题。（**队头阻塞**（**Head-of-line blocking**或缩写为**HOL blocking**）在[计算机网络](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%BD%91%E7%BB%9C" \o "计算机网络)的范畴中是一种性能受限的现象。它的原因是一列的第一个数据包（队头）受阻而导致整列数据包受阻）。HTTP/1.0不支持文件断点续传，HTTP/1.0每次传送文件都是从文件头开始，即0字节处开始。

#### HTTP1.1

HTTP 1.1支持持久连接（HTTP/1.1的默认模式使用带流水线的持久连接），在一个TCP连接上可以传送多个HTTP请求和响应，减少了建立和关闭连接的消耗和延迟。但每个单独的网页文件的请求和应答仍然需要使用各自的连接。HTTP 1.1还允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，但服务器端必须按照接收到客户端请求的先后顺序依次回送响应结果。在HTTP 1.1中增加Host请求头字段后，WEB浏览器可以使用主机头名来明确表示要访问服务器上的哪个WEB站点，HTTP 1.1还提供了与身份认证、状态管理和Cache缓存等机制相关的请求头和响应头。

#### HTTP2.0

**多路复用 (Multiplexing)**

多路复用允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息。在 HTTP/1.1 协议中浏览器客户端在同一时间，针对同一域名下的请求有一定数量限制。超过限制数目的请求会被阻塞。而 HTTP/2 的多路复用(Multiplexing) 则允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息。

**二进制分帧**

在二进制分帧层中， HTTP/2 会将所有传输的信息分割为更小的消息和帧（frame）,并对它们采用二进制格式的编码。

**首部压缩（Header Compression）**

### **session和cookie**

为什么要使用session和cookie这个话题就要从HTTP状态协议的无状态性开始说起了。**无状态协议是指协议对事物处理没有记忆能力**。为了解决HTTP传输大量重复信息内容的问题，cookie和session就登场了，它们可以为用户保存状态。

#### **Cookie**

cookie是通过客户端保持状态的解决方案。从定义上说，cookie就是服务器发送给客户端的特殊信息，而这些信息以文本文件的方式存放在客户端，然后客户端每次向服务器发送请求的时候都会带上这些特殊的信息。

当用户使用浏览器访问一个支持cookie的网站的时候，会有如下步骤：

1、用户会提供包括用户名在内的个人信息并且提交至服务器

2、服务器在向客户端会传相应的超文本的同时，发回这些个人信息。当然这些信息并不是存放在HTTP响应体（Response Body）中的，而是存放在HTTP响应头（Response Header）中的

3、当客户端浏览器收到来自服务器的响应之后，浏览器会将这些信息存放在一个统一的位置

4、之后，客户端再向服务器发送请求的时候，都会把相应的cookie再次返回至服务器，而这次，cookie信息则存放在HTTP请求头中了。

#### **Session**

session是相对于cookie的另外一个解决方案，它是通过服务器来保持状态的。session指的是服务器为客户端所开辟的存储空间，在其中保存的信息就是用于保存状态的。

首先，session是服务器端程序运行的过程中创建的，不同语言实现的应用程序有不同创建session的方法。在创建了session的同时，服务器会为该session生成唯一的sessionId，而这个sessionId被创建了之后，就可以调用session相关的方法往session中增加内容了，**而这些内容只会保存在服务器中，发送到客户端的只有sessionId**。当客户端再次发送请求时，会将这个sessionId带上，服务器收到请求之后就会根据sessionId找到对应的session，从而再次使用。这样的一个过程，让用户的状态得以保持。

### Post和get

GET产生一个TCP数据包；POST产生两个TCP数据包。

对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

### HTTPS

HTTPS其实是由两部分组成的：**HTTP+TLS/SSL**，即HTTP下加入TLS/SSL层，HTTPS的安全基础就是TLS/SSL。服务端和客户端的信息传输都会通过TLS/SSL进行加密，所以传输的数据都是加密之后的数据。TLS的前身就是SSL协议，因此没有特别说明TLS/SSL说的都是同一个东西。

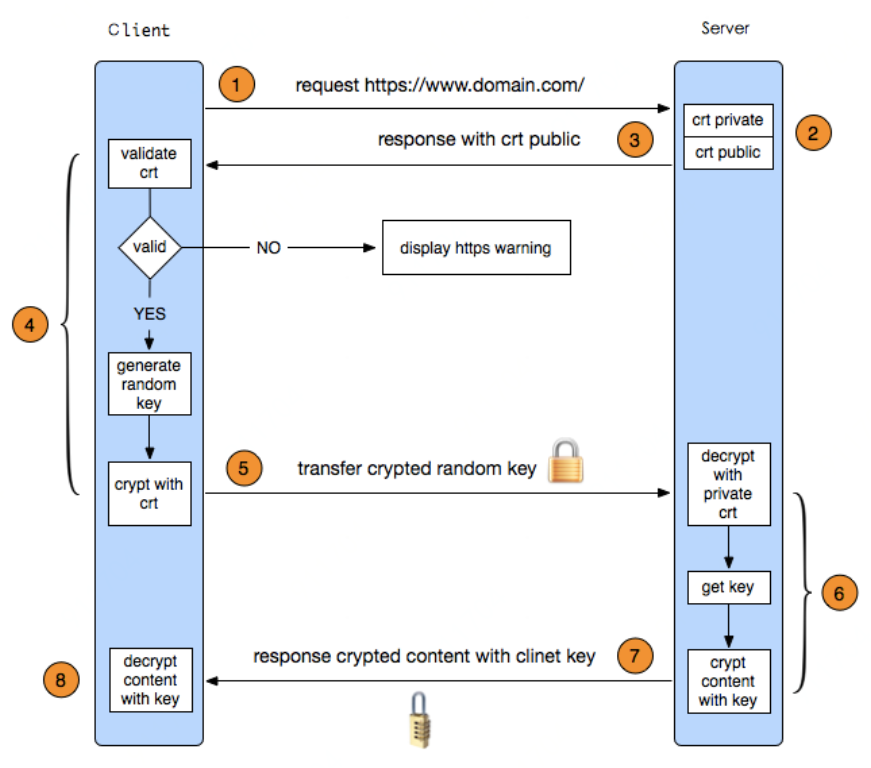
https的功能：

1、内容加密。浏览器到百度服务器的内容都是以加密形式传输的，中间者无法直接查看原始内容

2、身份认证。保证用户访问的是百度服务，即使被DNS劫持到了第三方站点，也会提醒用户没有访问百度服务

3、数据完整。防止内容被第三方冒充或篡改

**HTTPS工作原理**



**1、客户端发起HTTPS请求**

这个过程中客户端会发送一个密文族给服务端，密文族是浏览器所支持的加密算法的清单。

**2、服务端配置**

**采用HTTPS协议的服务器必须要有一套数字证书**，可以自己制作，也可以向组织申请。区别就是自己颁发的证书需要客户端验证通过才可以继续访问，而使用受信任的公司申请的证书则不会弹出提示页面。

这套证书其实就是一对公钥和私钥，可以这么理解，公钥就是一把锁头，私钥就是这把锁的钥匙，锁头可以给别人对某个东西进行加锁，但是加锁完毕之后，只有持有这把锁的钥匙才可以解锁看到加锁的内容。

**3、传送证书**

这个证书其实就是公钥。

**4、客户端解析证书**

这部分工作是由客户端的TLS来完成的，首先会验证公钥是否有效，如果证书没有问题，那么就生成一个随机值，然后用证书对该随机值进行加密。

**5、传送加密信息**

这部分传送的是用证书加密后的随机值，目的就是让服务端得到这个随机值，然后客户端和服务端的通信就可以通过这个随机值来进行加密和解密了。

**6、服务端解密信息**

服务端用私钥解密后，得到了客户端传过来的随机值，**至此一个非对称加密的过程结束，看到TLS利用非对称加密实现了身份认证和密钥协商**。然后把内容通过该值进行对称加密。

**7、传输加密后的信息**

这部分是服务端用随机值加密后的信息，可以在客户端被还原。

**8、客户端解密信息**

客户端用之前生成的随机值解密服务端传送过来的信息，于是获取了解密后的内容，**至此一个对称加密的过程结束，看到对称加密是用于对服务器待传送给客户端的数据进行加密用的**。